运用变异测试的并行程序测试用例最小化算法

郑 炜 冯 晨 吴潇雪 黄月明 方靓芸
(西北工业大学软件与微电子学院 西安 710072)

摘 要 在并行程序测试中,测试输入和线程交互时序是影响并行错误检测的两个关键因素。以缩减并行错误检测的输入空间为目标,给出一种基于变异测试的测试用例最小化算法。首先对并行程序进行研究,选取与并行错误密切相关的9个变异算子,并以此为基础为待测程序生成多种变异体;采用JPF作为线程调度工具来执行测试用例,根据变异评分与平均时间成本对测试用例进行排序,在优化后的测试用例集中选取检测能力不重复的测试用例,从而得到面向并行错误检测的最小测试用例集。实验结果证明,该方法能有效减小测试用例集的规模,并大幅缩短运行时间,从而提高了并行程序的测试效率。

关键词 并行程序,变异测试,测试用例优化,测试用例最小化
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.017

Mutation Test Based Test Case Minimization for Concurrent Program

ZHENG Wei FENG Chen WU Xiao-xue HUANG Yue-ming FANG Jing-yun
(School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract In the test of concurrency program, test input and the timing of threads interaction are the two key factors that affect the concurrency bug detection. For narrowing input space of concurrency bug detection, this paper proposed an algorithm of test case minimization based on mutation test. It first studies on different concurrency programs and selects 9 mutation operators which can contribute to the generation of mutants to the testing programs. Then it chooses JPF as the detecting tool during the execution of programs, through which the test cases are sorted according to mutation score(MS) and average cost(AC). Finally it picks those test cases which can find out different bugs in the prioritized test cases. Experimental results show that the method can effectively reduce the size of test cases, shorten running time and improve the effectiveness of concurrency bug detection.

Keywords Concurrency program, Mutation test, Test case prioritization, Test case minimization

1 引言

并行错误较强的隐蔽性和破坏性给其检测带来了极大困难。并行程序线程的随机调度机制使得错误只有在特定的线程交互下才能暴露,因此对测试用例的使用提出了较高的要求:需要执行大量的测试用例以尽可能多地覆盖所有线程调度情况。但若每次测试都执行全部测试用例,则会造成大量的时间消耗。这要求测试人员找出一个最小测试用例集,使其在测试效果与原测试用例集提供相同覆盖度的情况下的综合开销最小,从而消除冗余用例,寻求有效用例。

本文第2节介绍相关研究;第3节对所设计的算法及相关理论给出详细的方法来获取测试用例优先级和选择步骤,最终从原始测试集中获取一个最小测试用例;第4节通过一个具体示例对给出的算法和应用进行详细说明;第5节给出实验验证数据与结果对比;最后总结全文并展望未来。

2 相关研究

测试用例最小化技术并不是一个新问题,贪心算法、启发式算法、线性规划、遗传算法等都曾被应用于该课题。在之前的工作中,许多研究者探索了关于测试用例优先级的一些技术,如Manjka Tyagi等人[1]基于故障检测率、故障检测和风险检测能力的百分比这3个因素,给出了获取回归测试用例优先级的方法;Gökçe N等人[2]提出通过聚类分析对测试用例进行优化;Quriques等人[3]研究了模型结构和测试用例优化的影响;R. Kavitha等人[4]根据故障的严重性对测试用例优先级进行了划分。但是这些研究并没有针对并行错误测试用例最小化来给出一个优化。

并行系统线程调度的特性使得它的空间复杂度和时间复杂度相对更高,进而使得其检测相对比较困难。在给定的输入条件下,先前的工作已经对并行错误的检测有了显著的成

到稿日期:2016-10-07 返修日期:2016-12-04 本文受国家自然科学基金青年基金(61402370)资助。
郑 炜(1975-),男,副教授,硕士生导师,主要研究方向为基于互联网+的智慧城市、云环境下的软件开发与验证、复杂系统软件的开发与验证等,E-mail:wzheng@nwpu.edu.cn;冯 晨(1994-),女,硕士,主要研究方向为软件测试;吴潇雪(1983-),女,博士,主要研究方向为软件测试、软件质量保证等。

效[11-13]，但软件测试总是面对大量输入的情况，不可能对所有输入进行测试。因为一个多线程的程序可以采取多种不同的输入，并以更多不同的交互来执行每个输入，由此造成的巨大输入空间和交互空间大幅降低了测试运行速度，之后的错误校验使得输入执行速度慢了 10 倍甚至 100 倍[10]，导致执行所有测试输入的代价过高而无法被接受。同时，即使是同样的输入，由于线程调度的原因，也不一定会得到相同的结果。

针对这种情况，本文通过对基于变异测试的研究来优化测试用例并选择出它的最小测试集。变异测试是一种面向缺陷的单元测试技术，利用变异体来衡量测试套件是否充足的构想最初是由 DeMillo[14] 和 Hamlet[15] 提出的[16]。在并行程序中，运用变异测试的方法，根据被测程序特征设计变异算子，而变异算子在符合语法的前提下仅与原被测程序有微小的区别。根据变异算子生成相对应的变异体，识别出等价变异体的过程即为检测出错误的过程。若已有的测试用例不能消除所有的非等价变异体，则需要添加新的测试用例。因而在并行错误测试中，需找到一个测试用例综合开销最小的用例集，而非一个用例数最少的用例集，才能真正对测试用例集进行优化并提高测试效率。

在之前的工作中，一般是根据测试目标中的测试需求来给出相应的测试用例，并将所有测试用例组成一个测试集，以此作为初始用例集[17]；然后，采用贪心算法、启发式算法或者整数规划来对该测试用例集进行最小化，通过此过程去除冗余测试用例[5-6]。但这种方法并没有对并行程序错误检测产生特别好的效果，而且存在一定的缺点，如它的效果取决于最开始产生的测试用例集。

3 面向并行错误的测试用例最小化算法

3.1 并行程序变异算子的设计

目前，已经有一些研究针对并行系统的变异测试技术做出了改进，如 Bradbury J S 等人[7-8]基于 Java 并行错误模式和特性对其提出了 27 种变异算子。我们根据之前的研究对并行系统进行变异算子设计。

由一般并行错误类型可知，死锁和原子性违背是并行错误中最为典型的两类错误。通过研究这两类错误的特点和修复方法，选取了 9 种与之紧密相关的变异算子，如表 1 所列。

表 1 与线程和同步相关的变异算子

MXT	方法修改(如 wait()，sleep()，join()和 await()方法)：Modify Method-X Time (wait()，sleep()，join()和 await())
RTXC	删除线程方法：Remove Thread Method-X Call (wait()，join()，sleep()，yield()，notify()，notifyAll())
RCXC	删除并行机制方法(如锁、信号量、障碍等)：Remove Concurrency Mechanism Method-X Call
RNA	以 Notify()方法来代替 NotifyAll()方法：Replace NotifyAll() with Notify()
RJS	以 Sleep()方法来代替 Join()方法：Replace Join() with Sleep()
RSK	从方法中删除同步关键字：Remove Synchronized Keyword from Method
RSB	删除同步块：Remove Synchronized Block
MSP	修改同步块参数：Modify Synchronized Block Parameter
ESP	交换同步块参数：Exchange Synchronized Block Parameters
MXT	修改方法(如 wait()，sleep()，join()和 await())：Modify Method-X Time

这 9 种变异算子能较全面地展示并行错误的特性。同时，两种最普遍的并行错误模式——线程和同步通过这 9 种变异算子也能得到体现。其余 18 种变异算子对研究内容的影响不大，不再赘述。

3.2 算法

在测试用例优先级评定方面，从变异评分(MS)和平均成本(AC)两个方面来评价。给定一个程序 P 和一个测试集 T，MS 和 AC 的定义如下。

定义 1(变异评分(Mutation Score，MS)) 变异评分是一个测试中所检测出的变异算子占整个程序中变异算子总数的比例，在给定程序 P 和测试用例集 T 的情况下，MS 的计算方式如式(1)所示：

$$MS(P,T)=\frac{K_m}{T_m-E_m} \tag{1}$$

其中，K_m 表示被程序发现并消除的变异算子的数目，T_m 表示程序中变异算子的总数，E_m 表示发现的等价变异体的数目。变异评分用来评估一个测试集检测错误的能力，其理想状态为评分 1，表示该测试集 T 下的变异算子能检测出程序 P 中的所有错误。

定义 2(平均成本(Average Cost，AC)) 平均成本是指一个测试用例分别以不同变异体运行时的平均执行时间，其计算公式如式(2)所示：

$$AC(P,T)=\frac{C_1+C_2+\cdots+C_n}{n} \tag{2}$$

其中，n 表示所有变异体的数量；C_i 表示对一个变异体 M_i 执行测试集 T 中的一个测试用例 t 的实际运行时间。

算法伪代码如下。

```
输入：原始测试用例集 T，变异算子集 O(表 1 给出的 9 个变异算子)，
      待测程序 P
算法流程：
Begin
  M ←empty;//程序 P 的变异体集
  O′←empty;//从 O 中选择的程序 P 可用的变异算子集
  T′←empty;//优化后的最小测试用例集
  for(each o in O){
//步骤 1：获取程序 P 可用的变异算子 O′
if(MapOpt(P,o));//分析变异算子 o 对程序 P 的可用性
    AddMutOpt(O′,o);//将可用变异算子 o 加入 O′
  }
  If (O′!=empty){
//步骤 2：生成程序 P 的变异体集 M
  for(each o in O′){
      AddVariant(M,Mutate(P,o));
  }
//步骤 3：执行测试用例并标记执行结果
for(each t in T){
    for(each m in M){
      if(JPFExecute(m,t)==JPFExecute(P,t)){
        Mark(t,"S");//若原程序与变异体的执行结果相同，则标
                    为"S"
      else
        Mark(t,"F");//若原程序与变异体的执行结果不相同，则
                    标为"F"
```

```
        }
    }
//步骤 4：计算每个测试用例的 MS 值和 AC 值
    Mark(t,MS(t));//采用式(1)计算测试用例 t 的 MS 值，并赋给 t
    Mark(t,AC(t));//采用式(2)计算测试用例 t 的 AC 值，并赋给 t
    Mark(t,IDsofDetectedMutants(t)); //标记每个测试用例 t 所发
                                     现的变异体 ID
  }
//步骤 5：排序、生成最小测试用例集 T'
DesSortByMS(&T);//将 T 按照 MS 倒序排序
  AscSortSameMSByAC(&T));//将相同 MS 值的测试用例按照
                         AC 顺序排序
    RemvDupTCofSameMS(&T);//对于 MS 相同的测试用例，若所
                          发现的变异体也完全相同，则
                          只保留 AC 最小的一个，将其他
                          测试用例从 &T 中移除
    T'←T;
  Return T'
  End
输出：T'
```

4　生成最小测试用例的过程实例

以并行程序“Aireline. java”为例，详细说明使用该算法生成一个最小测试用例的过程。“Aireline. java”是测试工具 JPF 库的一个并行程序示例，拥有两个输入参数，分别为{numOfTickets}和{numThreads}。针对该程序，自动生成 6 个不同的测试输入组：根据实际意义，变量 numOfTickets 的取值范围为 0～100，变量 numTreads 的取值范围为 1～5 的整数。生成的测试用例如表 2 所列。

表 2　Aireline. java 测试的输入

编号	T_1	T_2	T_3	T_4	T_5	T_6
输入	48,4	92,2	70,1	69,2	2,2	20,2

步骤 1　选择合适的变异算子。

根据源代码特性，在程序中选取了 7 个可被使用的变异算子：MXT，RTXC，RJS，RSK，RSB，MSP，ESP。

步骤 2　生成变异体。

根据 7 个变异算子将对应生成的变异体命名为：Airline_MXT，Airline_RTXC，Airline_RJS，Airline_RSK，Airline_RSB，Airline_MSP，Airline_ESP。

步骤 3　执行测试用例并获取其结果。

以 JPF 作为检测工具来实现所有可能的并行调度[9]，使每个测试用例都在原程序中执行一次。将实验在原始程序执行的结果视为期望结果。执行变异体程序并将结果与原始程序进行对比，若与期望结果相同，则将该变异体对应的测试用例项标记为 S(Successful)，否则将其标记为 F(Fail)。示例程序的执行结果如表 3 所列。

表 3　测试用例的执行结果

编号	T_1	T_2	T_3	T_4	T_5	T_6
Airline_MXT	F	F	S	F	F	F
Airline_RTXC	F	F	S	F	S	F
Airline_RJS	S	F	S	S	S	F
Airline_RSK	S	F	S	S	S	F
Airline_RSB	S	S	S	S	F	S
Airline_MSP	F	F	S	F	S	F
Airline_ESP	F	F	S	F	S	F

步骤 4　计算 MS 值和 AC 值。

根据执行结果以及前文的计算公式可以得出各个测试用例的 MS 值，结果如表 4 所列。

表 4　测试用例的 MS 值

编号	MS
T_1	0.5714285714
T_2	0.8571428571
T_3	0
T_4	0.5714285714
T_5	0.2857142857
T_6	0.8571428571

表 5 列出了“Aireline. java”不同变异体的每个测试用例的实际执行时间。根据式(2)可以计算出其各自的 AC 值，结果如表 6 所列。

表 5　不同变异体测试用例的执行代价/s

	Airline_MXT	Airline_RTXC	Airline_RJS	Airline_RSK	Airline_RSB	Airline_MSP	Airline_ESP
T_1	0.6089999676	0.5469999313	0.6089999676	0.5130000114	5.8190000057	0.6860001087	0.9450001031
T_2	0.5459997654	0.5720000267	0.5410001282	0.5399999619	5.2910001278	0.6790001392	0.8720333333
T_3	0.5620000362	0.5239999294	0.5299999719	0.5099999914	0.8960001469	0.8029999733	0.5029879458
T_4	0.5339999199	0.5220000744	0.5339999199	0.5390000343	2.1340000629	0.6380000114	0.6380000114
T_5	0.7820000649	0.7410001755	0.7739999294	0.5329999924	0.6940000057	0.6840000153	0.2490000431
T_6	0.5340001584	0.5370001793	0.5260000229	0.5180001259	1.2030000687	0.6070001125	0.3970054128

表 6　不同测试用例的 AC 值

编号	AC 值
T_1	1.389714299
T_2	1.291576212
T_3	0.618283999
T_4	0.791285719
T_5	0.636714318
T_6	0.617429440

步骤 5　根据 MS 值和 AC 值排序。

评价测试用例的错误检测能力的第一要素是 MS 值。基于表 4 的 MS 值对各个测试用例进行降序排序，得出该测试集 T 中各个测试用例的优先级为 $T_2->T_6->T_1->T_4->T_5->T_3$。

从表 4 中可以看出，T_2 的 MS 值与 T_6 的相同，同时 T_1 与 T_4 也具有相同的 MS 值。为了进一步优化测试和节省测试成本，该算法同时也考虑了不同的执行时间可能会产生的效果。因此根据表 6 的 AC 值，将各测试用例的优先级进一步优化为：$T_6->T_2->T_4->T_1->T_5->T_3$。新生成的序列中，$T_6$ 的优先级最高，因此首先将其选入最小测试用例集 T' 中。由表 3 可知，T_6 发现的变异体为 Airline_MXT，Airline_RTXC，Airline_RJS，Airline_RSK，Airline_MSP 和

Airline_ESP。以 T_6 为基准，将其所发现的变异体加入至已发现变异体集 M' 中。在此基础上判断下一个测试用例 T_2 是否发现新的变异体，即 T_2 是否检测出不在 M' 中的变异体。若是，则将 T_2 加入到最小测试用例集 T' 中，否则舍弃 T_2。由表 3 得知 T_2 无法检测出新的变异体，因此将其舍弃。继续判断下一个测试用例 T_4，T_1，…，T_3，直至判定完所有的测试用例。至此可得到最小测试用例集 T'。在示例“Aireline.java”中，最终产生的测试用例集 T' 中包含两个测试用例（T_6，T_4），其余 4 个测试用例由于并未检测到新的变异体，因此被舍弃。

在后面的实验及结果分析中，将对得到的最小测试集与其他实验数据集一起进行评估讨论，通过对比分析得出结论。

5 实验结果分析

本文在实验时所采用的环境为：操作系统 Windows 7；处理器 Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz，内存 16GB；JPF 插件运行于 Eclipse(Luna Service Release 2(4.4.2))，JDK 版本为 1.8.0_45。

本文共选择 4 个程序作为实验数据集，其中 Aireline，LinkedList 以及 UnsortedTree 是来自于 JPF 所附带的测试程序集，另外一个较大规模的程序 LinkShareService 为作者所在课题组开发的大型智能社区平台系统中的一个后台服务。表 7 列出了实验采用的 4 个程序的代码行数、测试用例数以及采用表 1 列出的 9 个变异算子所产生的变异体数量。

表 7 实验中采用的程序数据

程序	代码行数	变异体数目	测试用例数
Aireline	38	7	6
LinkedList	179	6	20
UnSortedTree	122	38	20
LinkShareService	13803	293	573

实验通过分别计算和对比原始测试集与最小测试集所发现全部变异体消耗的时间成本 POC(Percentage Of Cost)和测试用例执行比率 POT(Percentage Of Executed Test Cases)来对最小测试集生成算法的效果进行评价，具体计算结果如表 8 所列。其中，POC-O 与 POT-T 分别表示原始测试集发现全部变异体所消耗的“时间成本比率”及“测试用例执行比率”；POC-M 与 POT-M 分别表示应用本文算法所产生的最小测试集发现所有变异体消耗的“时间成本比率”与“测试用例执行比率”。

表 8 4 个被测项目的耗费成本

Programs	POC-O	POC-M	POT-O	POT-M
Aireline	0.8843	0.2636	0.8333	0.3333
LinkedList	0.8138	0.1901	0.8015	0.1536
UnSortedTree	0.6512	0.1349	0.7134	0.1398
LinkShare Service	0.9342	0.0782	0.9901	0.0418

由表 8 可知，生成的最小测试集发现所有变异体所需要执行的测试用例及时间成本均远少于原始测试集，且随着程序规模的增加，这种差距更加明显。对于最小程序集 Aireline，其测试用例执行比率为原始测试用例集的33.33%，而其消耗的时间为执行原始测试集中全部测试用例时间的 26.36%。对于规模较大的程序 LinkShare Service，发现所有变异体所需测试用例数仅为原始测试用例的 4.18%，执行时间仅为原始时间的 7.82%。因此，在大规模程序中，针对并行错误的测试用例选择是极其重要的。运用本文所给出的算法可以保证在同等并行错误检测能力的情况下大幅降低测试成本，同时也为面向并行错误的测试用例选择提供了一种有效方法。

结束语 本文提出的方法先根据变异评分与平均时间成本对各测试用例进行排序，然后以优先级最高的测试用例为基准，按降序依次将各测试用例与基准进行对比，若无重复则将其加入生成的测试用例中。该方法是对当前并行系统错误检测技术的补充，并可与任何错误检测工具相组合来达到降低测试成本的目的。经实验证明，本文提出的最小测试用例方法有效地降低了并行错误检测成本，提高了并行系统测试效率。但本文研究只是面向并行错误测试用例最小化的一个初步尝试，还存在一些问题，比如如何将并行系统特征属性（如线程交互时序）作为测试用例优化的一个因素，这也是我们下一步要进行的研究工作。

参 考 文 献

[1] MANIKA T. An Approach for Test Case Prioritization Based on Three Factors[J]. Information Technology and Computer Science, 2015(4): 79-86.

[2] GÖKÇE N, BELLI F, EMINOV M, et al. Model-based test case prioritization using cluster analysis: a soft-computing approach [OL]. http://journals, tubitak. gov. tr/elektrik/abstract. htm?id=16001.

[3] OURIQUES J F S, CARTAXO E G, MACHADO P D L. Revealing influence of model structure and test case profile on the prioritization of test cases in the context of model-based testing[J]. Journal of Software Engineering Research & Development, 2015, 3(1): 1-28.

[4] KAVITHA R, SURESHKUMAR N. Test Case Prioritization for Regression Testing based on Severity of Fault[J]. International Journal on Computer Science & Engineering, 2010, 2(5): 1462-1466.

[5] CHEN T Y, LAU M F. A new heuristic for test suite reduction [J]. Information & Software Technology, 1998, 40(5): 347-354.

[6] CHEN T Y, LAU M F. A simulation study on some heuristics for test suite reduction[J]. Information & Software Technology, 1998, 40(13): 777-787.

[7] BRADBURY J S, CORDY J R, DINGEL J. Mutation Operators for Concurrent Java (J2SE 5.0)[C]// The Workshop on Mutation Analysis. IEEE Computer Society, 2006: 11.

[8] GLIGORIC M, ZHANG L, PEREIRA C, et al. Selective mutation testing for concurrent code[C]// International Symposium on Software Testing and Analysis. 2013: 224-234.

[9] DAN H, HIERONS R M. Semantic Mutation Analysis of Floating-Point Comparison[C] // IEEE Fifth International Conference on Software Testing, Verification and Validation. IEEE, 2012: 290-299.

[10] DENG D,ZHANG W,LU S. Efficient concurrency-bug detection across inputs[C]// Acm Sigplan International Conference on Ojbect Oriented Prgramming Systems Languages & Applications. ACM,2013:785-802.

[11] LUCIA B,CEZE L,STRAUSS K. ColorSafe:architectural support for debugging and dynamically avoiding multi-variable atomicity violations[J]. ACM Sigarch Computer Architecture News,2010,38(3):222-233.

[12] GAO Q,ZHANG W,CHEN Z,et al. 2ndStrike:toward manifesting hidden concurrency typestate bugs[J]. ACM SIGPLAN Notices,2012,47(4):239-250.

[13] MUZAHID A,QI S,TORRELLAS J. Vulcan:Hardware support for detecting sequential consistency violations dynamically[C]// 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE,2012:363-375.

[14] DEMILLO R A,LIPTION R J,SAYWARD F G. Hints on Test Data Selection:Help for the Practicing Programmer[J]. Computer,1978,11(4):34-41.

[15] HAMLET R G. Testing Programs with the Aid of a Compiler[J]. IEEE Transactions on Software Engineering,1977,SE-3(4):279-290.

[16] ANDREWS J H,BRIAND L C,Labiche Y. Is mutation an appropriate tool for testing experiments? [C]// International Conference on Software Engineering,2005 (ICSE 2005). IEEE,2005:402-411.

[17] ZHONG H,ZHANG L,MEI H. An experimental study of four typical test suite reduction techniques[J]. Information & Software Technology,2008,50(6):534-546.

(上接第 86 页)

构件系统动作序列以 Pi 演算自动验证工具 MWB 工具的格式载入并进行对比,以判断演化前后的构件系统是否满足一致性保持;最后通过具体实例的分析,表明本文的构件系统动态演化一致性验证方法不仅能验证演化前后构件系统的一致性,也能检测出演化前后构件系统的不一致性,确实是可行且有效的。

本文基于进程代数中弱互模拟理论定义了一致性准则,通过提取演化前后构件系统的外部行为来对其一致性进行验证,主要考虑了封闭简单场景模式下构件系统演化前后的一致性验证问题,因此本文未来的工作将主要考虑开放复杂多场景模式下构件系统演化一致性的验证,并就构件系统动态演化过程的一致性问题进行研究。

参 考 文 献

[1] 王映辉. 构件式软件技术[M]. 北京机械工业出版社,2012.

[2] YANG F Q,MEI H,LI K Q. Software Reuse and Software Component Technology[J]. Acta Electronica Sinica,1999,27(2):68-75. (in Chinese)
杨芙清,梅宏,李克勤. 软件复用与软件构件技术[J]. 电子学报,1999,27(2):68-75.

[3] PLASIL F,VISOVSKY S. Behavior protocols for software components[J]. IEEE Transactions on Software Engineering,2002,28(11):1056-1076.

[4] LUO Y,LI X Y,GUAN L W,et al. Study on Behaavior Consistency of System on Component Evolution[J]. Computer Science,2008,35(1):266-270. (in Chinese)
罗毅,李兴宇,关连伟,等. 构件演化中的系统行为一致性的研究[J]. 计算机科学,2008,35(1):266-270.

[5] SHEN L M,MA C,WANG T. Research on behavioral consistency of component dynamic evolution based on process algebra[J]. Application Research of Computers,2009,26(4):1345-1348. (in Chinese)
申利民,马川,王涛. 基于进程代数的构件动态演化行为一致性研究[J]. 计算机应用研究,2009,26(4):1345-1348.

[6] MA C,SHEN L M,WANG T. Behavior Consistency Verification Method Based on Component Dynamic Evolution[J]. Computer Engineering,2010,36(6):80-83. (in Chinese)
马川,申利民,王涛. 基于构件动态演化的行为一致性验证方法[J]. 计算机工程,2010,36(6):80-83.

[7] ZHOU Y,HUANG Y K,HUANG Z Q,et al. Towards an Approach of Consistency Verification for Online Software Evolution in Open Environments[J]. Journal of Software,2015,26(4):747-759. (in Chinese)
周宇,黄延凯,黄志球,等. 一种开放环境下软件在线演化一致性验证方法[J]. 软件学报,2015,26(4):747-759.

[8] VICTOR B,MOLLER F. The Mobility Workbench—a tool for the π-calculus[C]// International Conference on Computer Aided Verification. Springer Berlin Heidelberg,1994:428-440.

[9] MILNER R. Communicating and mobile systems:the pi calculus[M]. Cambridge University Press,1999.

[10] BERGSTRA J A,KLOP J W. Fixed point semantics in process algebras[J]. Stichting Mathematisch Centrum Informatica,1982:1-21.

[11] HU H Y,LV J,MA X X,et al. Study on Behavioral Compatibility of Components in Software Architecture Using Object-Oriented Paradigm[J]. Journal of Software,2006,17(6):1276-1286. (in Chinese)
胡海洋,吕建,马晓星,等. 面向对象范型体系结构中构件行为相容性研究[J]. 软件学报,2006,17(6):1276-1286.

[12] BERNARDO M,CIANCARINI P,DONATIELLO L. Architecting families of software systems with process algebras[J]. ACM Transactions on Software Engineering and Methodology (TOSEM),2002,11(4):386-426.

[13] DAI F,LI T,XIE Z W,et al. Towards an algebraic semantics of software evolution process models[J]. Journal of Software,2012,23(4):846-863. (in Chinese)
代飞,李彤,谢仲文,等. 一种软件演化过程模型的代数语义[J]. 软件学报,2012,23(4):846-863.